nº ord 2

A

1	2	3	Ejercicio 1	Eric Brandwein
B	R	B-		LU: 349/16

~~a. $\dfrac{}{\text{MultiSet}_\sigma <: \text{Set}_\sigma}$ (S-set-Multiset) → Para cada término en lo que se usa un Set$_\sigma$ se lo puede reemplazar por un MultiSet$_\sigma$. Veamos:~~

~~• Para $M \in N$, si se usa un Multiset$_\sigma$ para N, sigue siendo válido y resulta en el Bool que determina si hay por lo menos un elemento M en N.~~

~~•~~

En realidad $\text{Set}_\sigma <: \text{MSet}_\sigma$ sii operaciones de Set $\subseteq$ operaciones de mSet.

Por lo tanto pensar en esto no tiene sentido.

a. $\dfrac{}{\text{set}_\sigma <: \text{Multiset}_\sigma}$ (S-set-Multiset) → Para cada término en el que se use un Multiset$_\sigma$ se lo puede reemplazar por un set$_\sigma$. Veamos:

• Para $\#(M,N)$, si $\Gamma \triangleright N : \text{set}_\sigma$, el término resolverá a 1 o 0. ✓ ok.

• Para $M \setminus \{N\}$, si $\Gamma \triangleright M : \text{set}_\sigma$, el término resolverá a M sin el término eliminado.
✓

$\dfrac{\sigma <: \tau}{\text{Multiset}_\tau <: \text{Multiset}_\sigma}$ (S-Multiset) → Para cada término en el que se use un Multiset$_\sigma$ se lo puede reemplazar por un Multiset de un supertipo. Veamos:

• Para $\#(M,N)$ un Multiset$_\tau$ puede contener valores de tipo σ, y entonces preguntar las apariciones de un término de tipo σ tiene sentido. ¿son el mismo?

• Para $M \setminus \{N\}$, de nuevo, un Multiset$_\tau$ puede contener términos del tipo σ, y entonces eliminar un término de tipo σ tiene sentido.

$\dfrac{\sigma <: \tau}{\text{set}_\tau <: \text{set}_\sigma}$ (S-Set) → Las justificaciones son similares al caso anterior, pero con $M \in N$ en vez de $\#(M,N)$.

Tanto Multiset como set son contravariantes

b. Este nuevo Término hace que la regla (S-Set-Multiset) ya no sea válido, ya que no siempre se le puede agregar un elemento a un Set σ, por el hecho de tener como máximo un elemento de un mismo valor. ~~Como el Término M+N modifica el término a M en vez de devolver un nuevo valor,~~ ~~no podemos crear~~ Como el Término no devuelve un objeto nuevo, ~~no~~ sino que modifica a M, no se podría "convertir" al M a un Multiset si ~~[illegible]~~ quiere un set.

nerd 2 Broudevein

a-Bien b-Mal

Ejercicio 2

R

a. • console.log(o2.x) ⇝ 60. ✓

Esto es porque se llamó a la función y de o1 pero con contexto o2, y por lo tanto this en ese contexto era o2.

• console.log(o1.x) ⇝ 10. ✓

Por la misma razón de antes, el valor de o1.x no cambió.

• console.log(o2.z) ⇝ 30. ✓

El valor de o2.z no es encontrado en o2, y entonces se lo va a buscar al prototype. ✓

• console.log(o1.z) ⇝ 30. ✓

Como el prototype de tanto o1 como o2 es Object, cuando se cambió z en el prototype de o2 también se lo hizo en o1.

• console.log(o3.x) ⇝ undefined.

En ningún momento se definió el valor de x ni en o3 ni en ninguno de los objetos pertenecientes a su cadena de prototypes. ✓

• console.log(o3.z) ⇝ 40.

El ~~prototy~~ prototype de o3 es G.prototype. Como z está definido en G.prototype, y o3 no tiene definido z, o3.z es el valor de z en G.prototype.

→ hay elementos no toma ningún parámetro X false

b. Vacio $\stackrel{def}{=}$ [hoyElementos = $\varsigma(x)$ ~~[illegible]~~ x.contElementos>0,

contElementos = 0, HayElementos = false

elementos = []

~~agregar = ς(x) λ(e) x.elementos.append(e);~~

~~sacar = ς(x) λ(e) x.elementos,~~

~~agregar = ς(x1) λ(e) x1.new(contElementos+1, x1.elementos ∪ e),~~

~~sacar = ς(x) λ(e) x.new(contElementos-1, x.elementos/{e}),~~

~~pertenece = ς(x) λ(e) e ∈ x.elementos,~~

new = $\varsigma(x_1)$ λ(cElem)(elems) X Mal

[contElementos = cElem,

elementos = elems, pertenece = $\varsigma(x_2)$ x_1.pertenece(x_2),

hoyElementos = $\varsigma(x_2)$ x_1.hoyElementos(x_2),

sigue

$agregar = \varsigma(x_2)$ ~~x_1.agrego~~ ~~$\lambda(e)$~~ x_1.agregar(x_2),

$sacar = \varsigma(x_2)\ x_1$.sacar$(x_2)$],

agregarX $= \varsigma(x_1)\ \lambda(x_2)\ \lambda(e)\ x_1$.new($x_2$.contElementos+1, x_2.elementos $\cup\, e$),

sacar $= \varsigma(x_1)\ \lambda(x_2)\ \lambda(e)\ x_1$.new($x_2$.contElementos−1, x_2.elementos $\setminus\, e$),

pertenece $= \varsigma(x_1)\ \lambda(x_2)\ \lambda(e)$ ~~x~~ $e \in x_2$.elementos]

→ ? NO tenes un conj. definido

→ está mal esto. Tenes que redefinir los métodos del objeto que devolvés. Por ejemplo, O.agregar(x) devuelve un objeto al que si le preguntás si x pertenece devuelve True y sino devuelve O.pertenece.

nº ad 2
Braudwein

Ejercicio 3

B-

a - relacion(+A, +B, -R)

~~relacion([], _, []).~~
~~relacion(_, [], []).~~
~~relacion~~

relacion(A, B, Relacion) :-
    productoCartesiano(A, B, Producto),
    sublista(Relacion, Producto).

Donde asumo que sublista, ubicado en la guía práctica 6, ejercicio 5, genera las sublistas no necesariamente consecutivos, y donde productoCartesiano(+A, +B, -P) se define como:

así son consecutivas las del ej. de la práctica.

productoCartesiano([], _, []).
productoCartesiano([A|As], B, Producto) :-
    combinar(A, B, Head), productoCartesiano(As, B, Tail),
    concatenar(Head, Tail, Producto).

Donde concatenar es el predicado de la guía 6, ejercicio 4, y donde combinar(+Elemento, +B, -C) se define como:

combinar(_, [], []).
combinar(Elemento, [B|Bs], C) :-
    combinar(Elemento, Bs, Cs), C = [(Elemento, B), Cs].

es fea esta notación

b. funcion(+D, +I, -F)
funcion([ElemD|Ds], Imagen, F) :-
    member(ElemI, Imagen), funcion(Ds, Imagen, FT),
    F = [(ElemD, ElemI) | FT].
funcion([], _, []).

siempre instancia ElemI con el mismo elemento de la Imagen

nº ord 2 Preguntas Teóricas Eric Bronsvein

a. I) Verdadero, si unificaran $P(g(a,x),x,f(a))$ con $P(x,y,y)$, ~~...~~ unificaría con ~~...~~ $g(a,x)$, que a su vez unificaría con $g(a,g(a,x))$, que a su vez unificaría con $g(a,g(a,g(a,x)))$, etc. Como los literales deben ser finitos, no pueden unificar, y entonces no podemos avanzar en la resolución.

No: las variables x en las cláusulas son distintas. Deben renombrarse

II) Falso. La forma normal de Skolem ~~...~~ de la proposición es

$\forall x \forall y (P(x,f(x,y)) \wedge \neg P(y,f(x,y)))$

Ok

c. Como vimos en la teórica, la definición original del not es:

```
not(G) :- call(G), !, fail.
not(G).
```

El cut (!) corta el árbol SLD. ~~y entonces ...~~ Si tenemos un G que dé true en algún momento, en el caso redefinido se pasaría a evaluar el fail, que fallaría, y entonces se pasaría a ~~...~~ evaluar la siguiente condición, que es not(G), la cual es siempre true. ~~...~~ Nunca se llegará al cut como ocurre en la definición original, y por lo tanto ~~...~~ nunca fallaría.

primero hace backtracking con las restantes soluciones. Podría no terminar

b. La clase Conjunto ya está definida por el objeto Vacío del punto 2b, ya que usando el new podemos crear conjuntos de los elementos que queramos.

Cómo se explica en la corrección de 2.b, la solución es incorrecta porque asume que tiene un tipo de datos conjuntos. Esto que sugiere aquí está bien.